```lua
-- CODING CONVENTIONS:
-- No globals. Line length: 80 chars or less.
-- Parse settings from a help string (see top of file).
-- This code does no run anything. Rather it is a module to be loaded
-- and run by e.g. rlgo.lua)
--
-- VARIABLE NAME CONVENTIONS:
-- Leading__upper_case : class
-- i.                  :  instance var
-- l. s                : reference to a library function
-- prefix _            : some internal function,variable.
--
-- TYPE HINT CONVENTIONS (where practical, on function arguments):
-- - t = table
-- - prefix s=string
-- - prefix n=num
-- - prefix is=boolean
-- - class names in lower case denote vars of that class
-- - suffix s denotes table of things
local l = require"lib"
local the = l.settings[[

RL.LUA : stings
(c)2022 Tim Menzies <timm@ieee.org> BSD(2clause).

USAGE:
  lua rlgo.lua [ -bFghksS [ARG] ]

OPTIONS:
 -b  --bins   discretization control       = 8
 -F  --Far    in "far", how far to seek    = .95
 -f  --file   data file = ../../data/auto93.csv
 -g  --go     start-up action              = pass
 -h  --help   show help                    = false
 -k  --keep   keep only these nums         = 512
 -M  --Min    stop at m                    = 10
 -p  --p      distance coefficient         = 2
 -s  --seed   random number see            = 10019
 -S  --Some   in "far", how many to search = 10000
]]
local RL    = {About={}, Data={}, Row={},Col={},Xy={},Xys={},the=the}
local About= RL.About -- factory for making columns
local Data = RL.Data  -- store rows, and their column summaries
local Row  = RL.Row   -- stores one row.
local Col  = RL.Col   -- summarize 1 column. Has 2 roles-- NOMinal,RATIO for syms,nums
local Xy   = RL.Xy    -- summarize two columns from the same rows
local Xys  = RL.Xys   -- Manger for sets of "Xy"s.

-- I considered splitting Col into two (one for NOMinals and one for
-- RATIOs).  But as shown in Col (below), one of those two cases can usually be
-- handled as a one-liner. So the benefits of that reorg is not large.
--
-- To save memory, Rows are created by the firsT Data
-- that sees a record, then shared across every other clone  of the datav. Rows
-- hold a pointer to its creator Data object. Hence, that first data Data can be
-- used to store information about the entire data spaces (e.g. the max and min
-- possible values for each columns).  This makes certain functions easier like,
-- say, distance).
--       ____ ___  ____ _  _ ___
--       |__| |__] |  | |  |  |
--       |  | |__] |__| |__|  |
--
-- Factory for making columns.
function About.new(sNames)
  return About._cols({names=sNames, all={}, x={}, y={}, klass=nil},sNames) end

-- How to recognize different column types
local _is={
  nom   = "^[a-z]",  -- ratio cols start with uppercase
  goal  = "[!+-]$",  -- !=klass, [+,-]=maximize,minimize
  klass = "!$",       -- klass if "!"
  skip  = ":$",       -- skip if ":"
  less  = "-$"}        -- minimize if "-"

-- Turn a list of column names into Col objects. If the new col is independent
-- or dependent or a goal attribute then remember that in i.x or i.y or i.klass.
function About._cols(i,sNames)
  for at,name in pairs(sNames) do
    local col = l.push(i.all, Col.new(name,at))
    if not name:find(_is.skip) then
      l.push(name:find(_is.goal) and i.y or i.x, col)
      if name:find(_is.klass) then i.klass=col end end end
  return i end

-- Update, only the non-skipped cols (i.e. those found in i.x and j.x.
function About.add(i,t)
  local row = t.cells and t or Row.new(i, t)
  for _,cols in pairs{i.x,i.y} do
    for _,col1 in pairs(cols) do
      Col.add(col1, row.cells[col1.at]) end end
  return row end
--
--       ____ ____ _ _ _
--       |__/ |  | | | |
--       |  \ |__| |_|_|
--
-- Hold one record
function Row.new(about,t) return {_about=about,cells=t,cooked=l.map(t,l.same)} end

-- Everything in rows, sorted by distance to i.
function Row.around(i,rows)
  local fun = function(j) return {row=j, d=Row.dist(i,j)} end
  return l.sort(l.map(rows, fun), l.lt"d") end

-- Recommend sorting i before j (since i is better).
function Row.better(i,j)
  i.evaled,j.evaled= true,true
  local s1,s2,d,n,x,y=0,0,0,0
  local ys,e = i._about.y,math.exp(1)
  for _,col in pairs(ys) do
    x,y= i.cells[col.at], j.cells[col.at]
    x,y= Col.norm(col,x), Col.norm(col,y)
    s1 = s1 - e^(col.w * (x-y)/#ys)
    s2 = s2 - e^(col.w * (y-x)/#ys)
  end
  return s1/#ys < s2/#ys end

-- Distance
function Row.dist(i,j)
  local d,n,x,y,dist1=0,0
  local cols = cols or i._about.x
  for _,col in pairs(cols) do
    x,y = i.cells[col.at], j.cells[col.at]
    d   = d + col.infoGain+Col.dist(col,x,y)^the.p
    n   = n + col.infoGain end
  return (d/n)^(1/the.p) end
--       ____ ____ _
--       |    |  | |
--       |___ |__| |___
--
-- Summarize one column.
function Col.new(txt,at)
  txt = txt or ""
  return {n    = 0,                 -- how many items seen?
          at   = at or 0,         -- position ot column
          txt  = txt,           -- column header
          isNom= txt:find(_is.nom),
          w    = txt:find(_is.less) and -1 or 1,
          infoGain =1,
          ok   = true,              -- false if some update needed
          _has  = {}} end            -- place to keep (some) column values.

-- Create columns with particular roles.
function Col.ratio(...) local i=Col.new(...); i.isNom=false; return i end
function Col.nom(...)   local i=Col.new(...); i.isNom=true;  return i end

-- Update. Optically, repeat n times.
function Col.add(i,x,  n)
  if x ~= "?" then
    n = n or 1
    i.n = i.n + n
    if i.isNom then i._has[x] = n + (i._has[x] or 0) else
      for _ = 1,n do
        local pos
        if     #i._has  < the.keep     then pos=  1 + (#i._has)
        elseif l.rand() < the.keep/i.n then pos=l.rand(#i._has) end
        if pos then
          i.ok=false -- kept items are no longer sorted
          i._has[pos]=x end end end end end

-- Distance. If missing values, assume max distance.
function Col.dist(i,x,y)
  if x=="?" and y=="?" then return 1 end
  if i.isNom then return x==y and 0 or 1 else
    if     x=="?" then y = Col.norm(i,y); x=y<.5 and 1 or 0
    elseif y=="?" then x = Col.norm(i,x); y=x<.5 and 1 or 0
    else   x,y = Col.norm(i,x), Col.norm(i,y) end
    return math.abs(x-y) end end

-- Diversity: divergence from central tendency (sd,entropy for NOM,RATIO).
function Col.div(i)
  local t = Col.has(i)
  if not i.isNom then return (l.per(t,.9) - l.per(t,.1))/2.58 else
    local e=0
    for _,v in pairs(t) do if v>0 then e=e-v/i.n*math.log(v/i.n,2) end end
    return e end end

-- Sorted contents
function Col.has(i)
  if not i.isNom and not i.ok then table.sort(i._has); i.ok=true end
  return i._has end

-- Central tendency (mode,median for NOMs,RATIOs)
function Col.mid(i)
  if not i.isNom then return l.per(Col.has(i),.5) else
    local mode,most=nil,-1
    for k,v in pairs(i._has) do if v>most then mode,most=k,v end end
    return mode end end

-- Return num, scaled to 0..1 for lo..hi
function Col.norm(i,x)
  if i.isNom then return x else
    local has= Col.has(i) -- "a" contains all our numbers,  sorted.
    local lo,hi = has[1], has[#has]
    return hi - lo  < 1E-9 and 0 or (x-lo)/(hi-lo) end end

-- Map x to a small range of values. For NOMs, x maps to itself.
function Col.discretize(i,x)
  if i.isNom then return x else
    local has = Col.has(i)
    local lo,hi = has[1], has[#has]
    local b = (hi - lo)/the.bins
    return hi==lo and 1 or math.floor(x/b+.5)*b  end end
--       ___  ____ ___ ____
--       |  \ |__|  |  |__|
--       |__/ |  |  |  |  |
--
-- Holds n records
function Data.new(t) return {rows={}, about=About.new(t) } end

-- Update
function Data.add(i,t) l.push(i.rows, About.add(i.about,t)) end

-- Sort rows, then pretend you didn't
function Data.cheat(i)
  for j,row in pairs(l.sort(i.rows, Row.better)) do
    row.rank = l.rnd(100*j/#i.rows)
    row.evaled= false end
  i.rows = l.shuffle(i.rows) end

-- Replicate structure
function Data.clone(i,  t)
  local out = Data.new(i.about.names)
  for _,row in pairs(t or {}) do Data.add(out,row) end
  return out end

-- Discretize all row values (writing those vvalues to "cooked").
function Data.discretize(i)
  for _,row in pairs(i.rows) do
    for _,col in pairs(i.about.x) do
      local x = row.cells[col.at]
      if x~= "?" then
        row.cooked[col.at] = Col.discretize(col,x) end end end end

-- Diversity
function Data.div(i) return l.map(i.about.y, Col.div) end

function Data.infoGain(i)
  for n,rows in pairs(Data.leaves(i,3)) do
    for _,row in pairs(rows) do
      row.label= n end end
  for _,col in pairs(i.about.x) do
    col.infoGain = Xys.infoGain(Xys.bins(i.rows, col)) end end

-- Recursively bi-cluster one Data into sub-Datas.
function Data.leaves(i,depth)
  local stop   = the.Min
  local leaves = {}
  local function worker(rows, depth, rowAbove)
    if   depth <= 0 or #rows < 2*stop
    then l.push(leaves, rows)
    else local A,B,As,Bs = Data.half(i,rows,rowAbove)
         worker(As, depth-1, A)
         worker(Bs, depth-1, B) end end
  worker(i.rows, depth or 10)
  return leaves end

-- Split data according to distance to two  distant points A,B
-- To dodge outliers, don't search all the way to edge (see the.Far).
-- To speed things up:
-- - try to reuse a distant point from above (see rowAbove).
-- - only look at some of the rows (see the.Some).
-- - find distant points in linear time via
--   A=far(any()) and B=far(A).
function Data.half(i, rows,  rowAbove,      c)
  local some= l.many(rows, the.Some)
  local function far(row)
    return l.per(Row.around(row,some), the.Far).row end
  local As,Bs = {},{}
  local A= rowAbove or far(l.any(some))
  local B= far(A)
  local c= Row.dist(A,B)
  local function project(row)
    local a,b = Row.dist(row,A), Row.dist(row,B)
    return {row=row, x=(a^2 + c^2 - b^2)/(2*c)} end
  for n,rowx in pairs(l.sort(l.map(rows, project),l.lt"x")) do
    l.push(n < #rows/2 and As or Bs, rowx.row) end
  return A,B,As,Bs,c end

-- Load from file
function Data.load(sFilename,         data)
  l.csv(sFilename, function(row)
    if data then Data.add(data,row) else data=Data.new(row) end end)
  return data end

-- Central tendency
function Data.mid(i)
  local t={}
  for _,col in pairs(i.about.y) do t.n=#i.rows; t[col.txt] = Col.mid(col) end
  return t end

-- Return final best and first worst
function Data.best(i,  rows, rowAbove,stop,worst)
  stop = stop or the.Min
  rows = rows or i.rows
  if   #rows <= stop
  then return rows,worst
  else local A,B,As,Bs,c = Data.half(i, rows, rowAbove)
       if   Row.better(A,B)
       then return Data.best(i,As,A,stop,worst or Bs)
       else return Data.best(i,Bs,B,stop,worst or As) end end end
```

```lua
--       _  _ _  _
--        \/   \_/
--       _/\_   |
function Xy.new(str,at,num1,num2,nom)
  return {txt = str,
          at  = at,
          xlo = num1,
          xhi = num2 or num1,
          y   = nom or Col.nom(str,at)} end

function Xy.add(i,x,y)
  i.xlo = math.min(x, i.xlo)
  i.xhi = math.max(x, i.xhi)
  Col.add(i.y, y) end

function Xy.show(i)
  local x,lo,hi = i.txt, i.xlo, i.xhi
  if     lo ==    hi  then return l.fmt("%s == %s", x, lo)
  elseif hi ==  l.big then return l.fmt("%s > %s", x, lo)
  elseif lo == -l.big then return l.fmt("%s <= %s", x, hi)
  else                     return l.fmt("%s < %s <= %s", lo,x,hi) end end


-- Xys is a set of class methods that handle lists of "Xy"s.
function Xys.bins(rows,col)
  local n,xys = 0,{}
  for _,row in pairs(rows) do
    local x = row.cells[col.at]
    if x~= "?" then
      n = n+1
      local bin = Col.discretize(col,x)
      local xy  = xys[bin] or Xy.new(col.txt,col.at, x)
      Xy.add(xy, x, row.label)
      xys[bin] = xy end end
  local tmp={}
  for n,xy in pairs(xys) do l.push(tmp,xy) end
  xys = l.sort(tmp, l.lt"xlo")
  return col.isNom and xys or Xys._merges(xys,n^.5) end

function Xys.infoGain(xys)
  local n,out,all=0,0,Col.nom()
  for _,xy in pairs(xys) do
    for x,n in pairs(xy.y._has) do Col.add(all,x,n) end
    n   = n   + xy.y.n end
  for _,xy in pairs(xys) do out = out + xy.y.n/n * Col.div(xy.y) end
  return Col.div(all) - out end

-- While adjacent things can be merged, keep merging.
-- Then make sure the bins to cover &pm; &infin;.
function Xys._merges(xys0,nMin)
  local n,xys1 = 1,{}
  while n <= #xys0 do
    local xymerged = n<#xys0 and Xys._merged(xys0[n],xys0[n+1],nMin)
    xys1[#xys1+1]  = xymerged or xys0[n]
    n = n + (xymerged and 2 or 1) -- if merged, skip next bin
  end
  if   #xys1 < #xys0
  then return Xys._merges(xys1,nMin)
  else xys1[1].xlo = -l.big
       for n=2,#xys1 do xys1[n].xlo = xys1[n-1].xhi end
       xys1[#xys1].xhi = l.big
       return xys1 end end

-- Merge two bins if they are too small or too complex.
-- E.g. if each bin only has "rest" values, then combine them.
-- Returns nil otherwise (which is used to signal "no merge possible").
function Xys._merged(xy1,xy2,nMin)
  local i,j= xy1.y, xy2.y
  local k = Col.nom(i.txt, i.at)
  for x,n in pairs(i._has) do Col.add(k,x,n) end
  for x,n in pairs(j._has) do Col.add(k,x,n) end
  local tooSmall   = i.n < nMin or j.n < nMin
  local tooComplex = Col.div(k) <= (i.n*Col.div(i) + j.n*Col.div(j))/k.n
  if tooSmall or tooComplex then
    return Xy.new(xy1.txt,xy1.at, xy1.xlo, xy2.xhi, k) end end


-- ---------------------------------------------------------------------------
return RL
```